Introduction to Computational Physics

PHYS 250 (Autumn 2018) – Lecture 4

David Miller

Department of Physics and the Enrico Fermi Institute
University of Chicago

October 11, 2018

Outline

1. *Visualization methods and techniques*
   - Recall: random walks
   - Perspectives on visualization
   - Visualization libraries and concepts

Slow random diffusion

Last time, we determined that if the diffusion (i.e. walking speed) is also slow, such that the time derivative of the position probability distribution function, ρ, is approximately linear with respect to time and $\rho(x, t + \Delta t) - \rho(x, t) \approx (\frac{\partial \rho}{\partial t})\Delta t$, then

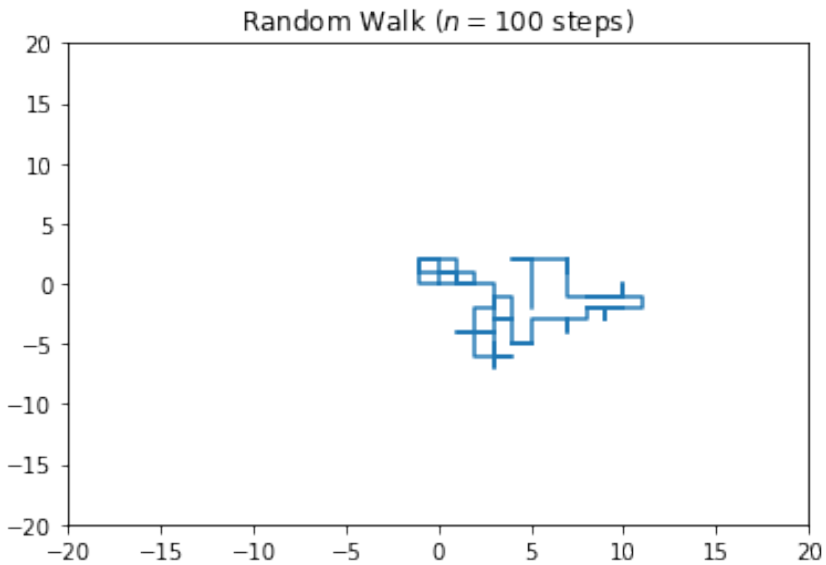
$$\frac{\partial \rho}{\partial t} = \frac{a^2}{2\Delta t}\frac{\partial^2 \rho}{\partial x^2}. \tag{1}$$

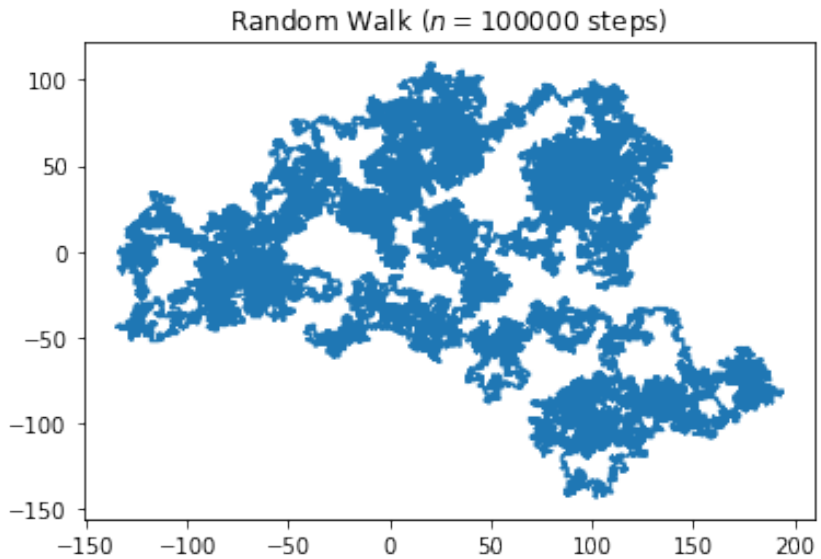
We observed that this **is** the diffusion equation Eq. **??** with $D = \dfrac{a^2}{2\Delta t}$.

The point is that we obtained an analytical description of a random walk via the diffusion equation under minimal assumptions: the probability distribution is broad and slowly varying compared to the size and time of the individual steps.

Visualizing the random walk

Even just in our examples on Tuesday, we took the time to visualize what we were doing:





These were fairly simple to setup:

```
plt.plot(walk[0],walk[1],label= 'Random walk')
plt.title("Random Walk ($n="+str(nsteps)+"$ steps)")
```

Taking a step back on plotting and visualization

All of the visualization tools that we will discuss are powerful enough for professional scientific work and are free or open source.

- Commercial packages such as Matlab, AVS, Amira, and Noesys produce excellent scientific visualization but are less widely available.
- Mathematica and Maple have excellent visualization packages as well, but they are not nearly as useful for with large numerical data sets.

The powerhouse is `matplotlib`

A very powerful library of plotting functions callable from within Python that is capable of producing publication quality figures in a number of output formats. It is, by design, similar to the plotting packages with MATLAB, and is made more powerful by its use of the `numpy` package for numerical work. In addition to 2-D plots, Matplotlib can also create interactive, 3-D visualizations of data.

An aside on philosophy

One of the absolute **best** parts about data analysis is making that *final, concise, precise figure* that clearly conveys the point of the results that you've been working on for so long.
But more than that, actually **looking at your data** is critical (as I showed clearly in the second lecture, I hope!).

- **visualizations may provide deep insights into problems by letting us see and *handle* the functions with which we are working**
- **visualization also assists in the debugging process**
- **visualizations are, simply, often just beautiful**

In thinking about ways to present your results, keep in mind that **the point of visualization is to make the science clearer and to communicate your work to others**. It follows then that you should make all figures as **clear, informative, and self-explanatory as possible**, especially if you will be using them in presentations without captions. This means labels for all curves and data points, a title, and labels on the axes.

Matplotlib documentation

https://matplotlib.org/index.html

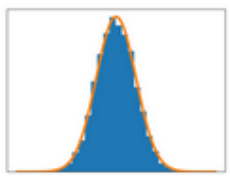
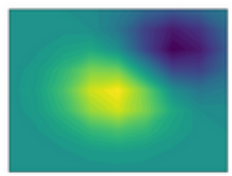
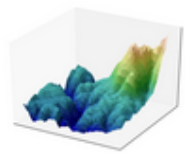
Matplotlib has fantastic documentation with nearly and infinite number of examples of how to build clear, useful, efficient data visualizations and figures.

matplotlib

Version 3.0.0

home | examples | tutorials | API | docs »

Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and IPython shells, the Jupyter notebook, web application servers, and four graphical user interface toolkits.

Matplotlib tries to make easy things easy and hard things possible. You can generate plots, histograms, power spectra, bar charts, errorcharts, scatterplots, etc., with just a few lines of code. For examples, see the sample plots and thumbnail gallery.

For simple plotting the `pyplot` module provides a MATLAB-like interface, particularly when combined with IPython. For the power user, you have full control of line styles, font properties, axes properties, etc, via an object oriented interface or via a set of functions familiar to MATLAB users.

"Hands-on"!

Today will be mostly going through examples, and doing hands-on exercises before we get back into the physics in a deep way next week.